Capítulo 1: Requisitos de software

Definición de un requisito de software

En su forma más básica, un requisito de software es una propiedad que debe ser exhibida por algo para resolver algún problema en el mundo real. Puede apuntar a automatizar parte de una tarea para que alguien apoye los procesos comerciales de una organización, corrija las deficiencias del software existente o controle un dispositivo, por nombrar solo algunos de los muchos problemas para los que las soluciones de software son posibles. Las formas en que funcionan los usuarios, los procesos comerciales y los dispositivos suelen ser complejas. Por extensión, por lo tanto, los requisitos de un software en particular suelen ser una combinación compleja de varias personas en diferentes niveles de una organización, y que de una forma u otra están involucradas o conectadas con esta característica del entorno en el que operará el software.

Los requisitos *funcionales* describen las funciones que debe ejecutar el software; por ejemplo, formatear un texto o modular una señal. A veces se les conoce como capacidades o características. Un requisito funcional también puede describirse como aquel para el que se puede escribir un conjunto finito de pasos de prueba para validar su comportamiento.

Los requisitos *no funcionales* son los que actúan para restringir la solución. Los requisitos no funcionales a veces se conocen como restricciones o requisitos de calidad. Pueden clasificarse además según sean requisitos de rendimiento, requisitos de mantenimiento, requisitos de seguridad, requisitos de confiabilidad, requisitos de seguridad, requisitos de interoperabilidad o uno de muchos otros tipos de requisitos de software (consulte Modelos y características de calidad en el KA de calidad del software).

Requisitos del sistema y requisitos del software En este tema, "sistema" significa una combinación interactiva de elementos para lograr un objetivo definido. Estos incluyen hardware, software, firmware, personas, información, técnicas, instalaciones, servicios y otros elementos de soporte.tal como lo define el Consejo Internacional de Ingeniería de Sistemas y Software (INCOSE).Los requisitos del sistema son los requisitos del sistema en su conjunto. En un sistema que contiene componentes de software, los requisitos de software se derivan de los requisitos del sistema.Esta KA define los “requisitos del usuario” de forma restringida, como los requisitos de los clientes o usuarios finales del sistema. Los requisitos del sistema, por el contrario, abarcan los requisitos del usuario, los requisitos de otras partes interesadas (como las autoridades reguladoras) y los requisitos sin una fuente humana identificable.

- Usuarios: este grupo comprende a aquellos que operarán el software. A menudo es un grupo heterogéneo que involucra a personas con diferentes roles y requisitos.
- Clientes: Este grupo comprende a aquellos que han encargado el software o que representan el mercado objetivo del software.
- Analistas de mercado: un producto de mercado masivo no tendrá un cliente encargado, por lo que a menudo se necesita personal de marketing para establecer lo que necesita el mercado y actuar como clientes proxy.
- Reguladores: muchos dominios de aplicación, como la banca y el transporte público, están regulados. El software de estos dominios debe cumplir con los requisitos de las autoridades reguladoras.
- Ingenieros de software: estas personas tienen un interés legítimo en beneficiarse del desarrollo del software, por ejemplo, reutilizando componentes en otros productos. Si, en este escenario, un cliente de un producto en particular tiene requisitos específicos que comprometen el potencial de reutilización de componentes, los ingenieros de software deben sopesar cuidadosamente su propia participación contra la de los clientes..

Capítulo 2: Diseño de software

En un sentido general, el diseño puede verse como una forma de resolución de problemas. Por ejemplo, el concepto de un problema perverso —un problema sin solución definitiva— es interesante en términos de comprender los límites del diseño. Varias otras nociones y conceptos también son de interés para comprender el diseño en su sentido general: objetivos, restricciones, alternativas, representaciones y soluciones (consulte Técnicas de resolución de problemas en Computing Foundations KA). El diseño de software es una parte importante del proceso de desarrollo de software. Para comprender el papel del diseño de software, debemos ver cómo encaja en el ciclo de vida del desarrollo de software. Por lo tanto, es importante comprender las principales características del análisis de requisitos de software, diseño de software, construcción de software, pruebas de software y mantenimiento de software.

1. El diseño arquitectónico (también conocido como diseño de alto nivel y diseño de nivel superior) describe cómo se organiza el software en componentes.
2. El diseño detallado describe el comportamiento deseado de estos componentes.

La abstracción es "una vista de un objeto que se enfoca en la información relevante para un propósito particular e ignora el resto de la información" [1] (ver Abstracción en Computing Foundations KA). En el contexto del diseño de software, dos mecanismos clave de abstracción son la parametrización y la especificación. La abstracción mediante parametrización se abstrae de los detalles de las representaciones de datos al representar los datos como parámetros con nombre. La abstracción por especificación conduce a tres tipos principales de abstracción: abstracción procedimental, abstracción de datos y abstracción de control (iteración).

Acoplamiento y cohesión . El acoplamiento se define como "una medida de la interdependencia entre módulos en un programa de computadora", mientras que la cohesión se define como "una medida de la fuerza de asociación de los elementos dentro de un módulo" [1].

Descomposición y modularización . Descomponer y modularizar significa que el software grande se divide en varios componentes con nombre más pequeños que tienen interfaces bien definidas que describen las interacciones de los componentes. Por lo general, el objetivo es colocar diferentes funcionalidades y responsabilidades en diferentes componentes.

La encapsulación y el ocultamiento de información significa agrupar y empaquetar los detalles internos de una abstracción y hacer que esos detalles sean inaccesibles para entidades externas.

Separación de interfaz e implementación . Separar la interfaz y la implementación implica definir un componente especificando una interfaz pública (conocida por los clientes) que está separada de los detalles de cómo se realiza el componente (ver encapsulación y ocultación de información arriba).

Suficiencia, integridad y primitividad. Lograr la suficiencia y la integridad significa asegurarse de que un componente de software capture todas las características importantes de una abstracción y nada más. La primitividad significa que el diseño debe basarse en patrones que sean fáciles de implementar.

Separación de preocupaciones . Una preocupación es un "área de interés con respecto al diseño de software" [8]. Una preocupación de diseño es un área de diseño que es relevante para una o más de sus partes interesadas. Cada vista de la arquitectura enmarca una o más preocupaciones. Separar las preocupaciones por puntos de vista permite a las partes interesadas centrarse en unas pocas cosas a la vez y ofrece un medio para gestionar la complejidad

Estilos arquitectónicos

Un estilo arquitectónico es “una especialización de tipos de elementos y relaciones, junto con un conjunto de restricciones sobre cómo se pueden usar” . Por tanto, se puede considerar que un estilo

arquitectónico proporciona la organización de alto nivel del software. Varios autores han identificado varios estilos arquitectónicos importantes:

- Estructuras generales (por ejemplo, capas, tuberías y filtros, pizarra)
- Sistemas distribuidos (por ejemplo, cliente-servidor, tres niveles, intermediario)
- Sistemas interactivos (por ejemplo, Modelo-Vista-Controlador, Presentación-Abstracción-Control)
- Sistemas adaptables (por ejemplo, microkernel, reflexión)
- Otros (por ejemplo, lotes, intérpretes, control de procesos, basados en reglas).

3.3 Patrones de diseño

Descrito de forma sucinta, un patrón es “una solución común a un problema común en un contexto dado”]. Si bien los estilos arquitectónicos pueden verse como patrones que describen la organización de alto nivel del software, se pueden usar otros patrones de diseño para describir detalles en un nivel inferior. Estos patrones de diseño de nivel inferior incluyen lo siguiente:

- Patrones de creación (por ejemplo, constructor, fábrica, prototipo, singleton)
- Patrones estructurales (por ejemplo, adaptador, puente, compuesto, decorador, fachada, peso mosca, proxy)
- Patrones de comportamiento (por ejemplo, comando, intérprete, iterador, mediador, recuerdo, observador, estado, estrategia, plantilla, visitante).

Principios generales de diseño de la interfaz de usuario

- *Capacidad de aprendizaje* . El software debe ser fácil de aprender para que el usuario pueda comenzar a trabajar rápidamente con el software.
- *Familiaridad del usuario* . La interfaz debe utilizar términos y conceptos extraídos de las experiencias de las personas que utilizarán el software.
- *Consistencia* . La interfaz debe ser coherente para que las operaciones comparables se activen de la misma forma.
- *Sorpresa mínima* . El comportamiento del software no debería sorprender a los usuarios.
- *Recuperabilidad* . La interfaz debe proporcionar mecanismos que permitan a los usuarios recuperarse de los errores.
- *Orientación al usuario* . La interfaz debe proporcionar comentarios significativos cuando se produzcan errores y proporcionar ayuda relacionada con el contexto a los usuarios.
- *Diversidad de usuarios* . La interfaz debe proporcionar mecanismos de interacción adecuados para diversos tipos de usuarios y para usuarios con diferentes capacidades (ciegos, deficientes visuales, sordos, daltónicos, etc.)

Capítulo 3: Construcción de software

1.1 Minimizar la complejidad

La mayoría de las personas tienen una capacidad limitada para mantener estructuras e información complejas en sus memorias de trabajo, especialmente durante largos períodos de tiempo. Esto demuestra ser un factor importante que influye en la forma en que las personas transmiten la intención a las computadoras y conduce a uno de los impulsores más fuertes en la construcción de software: minimizar complejidad. La necesidad de reducir la complejidad se aplica esencialmente a todos los aspectos de la construcción de software y es particularmente crítica para probar las construcciones de software. En la construcción de software, la complejidad reducida se logra enfatizando la creación de código que es simple y legible en lugar de inteligente. Se logra mediante el uso de estándares , diseño modular y muchas otras técnicas específicas . También está respaldado por técnicas de calidad centradas en la construcción.

1.2 Anticipar el cambio

La mayoría del software cambiará con el tiempo y la anticipación del cambio impulsa muchos aspectos de la construcción del software; Los cambios en los entornos en los que opera el software también afectan al software de diversas formas. Anticipar el cambio ayuda a los ingenieros de software a crear software extensible, lo que significa que pueden mejorar un producto de software sin interrumpir la estructura subyacente. La anticipación del cambio está respaldada por muchas técnicas específicas

1.3 Construcción para verificación

Construir para verificación significa construir software de tal manera que los ingenieros de software que escriben el software puedan encontrar fácilmente las fallas, así como los probadores y usuarios durante las pruebas independientes y las actividades operativas. Las técnicas específicas que apoyan la construcción para la verificación incluyen seguir los estándares de codificación para respaldar las revisiones de código y las pruebas unitarias, organizar el código para respaldar las pruebas automatizadas y restringir el uso de estructuras de lenguaje complejas o difíciles de entender, entre otras.

1.4 Reutilización

La reutilización se refiere al uso de activos existentes para resolver diferentes problemas. En la construcción de software, los activos típicos que se reutilizan incluyen bibliotecas, módulos, componentes, código fuente y activos comerciales listos para usar (COTS). La reutilización se practica mejor de forma sistemática, de acuerdo con un proceso repetible y bien definido. La reutilización sistemática puede permitir mejoras significativas en la productividad, la calidad y los costos del software. La reutilización tiene dos facetas estrechamente relacionadas: "construcción para reutilización" y "construcción con reutilización". El primero significa crear activos de software reutilizables, mientras que el segundo significa reutilizar activos de software en la construcción de

una nueva solución. La reutilización a menudo trasciende los límites de los proyectos, lo que significa que los activos utilizados se pueden construir en otros proyectos u organizaciones.

1.5 Normas de construcción

La aplicación de estándares de desarrollo internos o externos durante la construcción ayuda a lograr los objetivos de eficiencia, calidad y costo de un proyecto. Específicamente, las opciones de subconjuntos de lenguajes de programación permitidos y estándares de uso son ayudas importantes para lograr una mayor seguridad. Los estándares que afectan directamente los problemas de construcción incluyen

- métodos de comunicación (por ejemplo, estándares para formatos y contenidos de documentos)
- lenguajes de programación (por ejemplo, estándares de lenguaje para lenguajes como Java y C ++) * estándares de codificación (por ejemplo, estándares para convenciones de nomenclatura, diseño y sangría)
- plataformas (por ejemplo, estándares de interfaz para llamadas al sistema operativo)
- herramientas (por ejemplo, estándares de diagramación para notaciones como UML (Lenguaje de modelado unificado)).

Uso de estándares externos . La construcción depende del uso de estándares externos para lenguajes de construcción, herramientas de construcción, interfaces técnicas e interacciones entre Software Construction KA y otros KA. Los estándares provienen de numerosas fuentes, incluidas las especificaciones de interfaz de hardware y software (como Object Management Group (OMG)) y organizaciones internacionales (como IEEE o ISO). Uso de estándares internos. Las normas también pueden crearse sobre una base organizativa a nivel corporativo o para su uso en proyectos específicos. Estos estándares apoyan la coordinación de las actividades del grupo, minimizando la complejidad, anticipando el cambio y construyendo para la verificación

CAPITULO 4. PRUEBAS DEL SOFTWARE

Es una actividad que permite evaluar y mejorar la calidad del producto con el fin de detectar fallas y corregir errores.

Las pruebas del software consisten en verificar el comportamiento de un programa dinámicamente a través de un grupo finito de casos de prueba, debidamente seleccionados. Se ha ido cambiando la percepción de que las pruebas de software se realizan únicamente al final del proceso de creación de código fuente, siendo muy útil hacerlo en todas las etapas del desarrollo del software; esto permite corregir errores y detectar fallas de fondo, a tiempo.

FUNDAMENTOS DE PRUEBAS DE SOFTWARE

- Terminología relacionada con las pruebas
- Elementos clave
- Relación de las pruebas con otras actividades

NIVELES DE PRUEBA

- El objeto de la prueba
- Objetivos de la prueba

TECNICAS DE PRUEBA

- Pruebas basadas en la intuición y experiencia
- Técnicas basadas en las especificaciones
- Técnicas basadas en código
- Técnicas basadas en los errores
- Técnicas basadas en el uso
- Técnicas basadas en la naturaleza de la aplicación
- Seleccionando y combinación de técnicas

MEDIDAS DE LA PRUEBAS

- Evaluación de un programa mediante pruebas

· Evaluación de las pruebas realizadas

PROCESO DE PRUEBAS

· Consideraciones practicas

· Actividades de las pruebas

CAPITULO 5. MANTENIMIENTO DE SOFTWARE

El proceso de desarrollo de software debe satisfacer los requerimientos planteados, una vez en operación el proceso de cubrimiento de defectos, operación y cambio de ambiente debe darse en esta etapa, la fase de mantenimiento empieza con un periodo de garantía y de soporte post-implementación, pero el mantenimiento del software ocurre mucho antes.

Aunque la etapa de mantenimiento del software no ha tenido el grado de atención que se debe este tipo de desarrollo de software ya está empezando a cambiar ya que muchos errores graves han ocurrido por no prestarle la atención que se merece.

El mantenimiento de software se ha definido como el número total de actividades requeridas para proveer soporte efectivo al software, esto incluye un planeamiento efectivo antes. Durante y después de la implementación del software.

Aspectos Fundamentales en el mantenimiento del software:

El estándar IEEE/EIA 12207 define el mantenimiento como uno de los procesos principales en el ciclo de vida del software, el objetivo es modificar el software existente preservando su integridad también lo hacen en estos mismos términos la ISO/IEC 14764 este enfatiza en las entregas previas para la planeación del mantenimiento del software.

La necesidad de mantenimiento se da para garantizar que el software cumple satisfactoriamente con los requerimientos solicitados, este se aplica a cualquier desarrollo independiente del modelo de ciclo de vida utilizado, el mantenimiento se da en orden de alcanzar el desempeño adecuado y en el orden de:

- Corregir fallas.
- Improvisar el diseño.
- Implementar correcciones.

- Interfaces con otros sistemas.

Un numero de factores claves debemos tener presentes para asegurar el mantenimiento efectivo del software, es importante comprender que el mantenimiento de software nos proporciona una técnica única en los desafíos de administración para los ingenieros de software, podemos apreciar cómo se planean las liberaciones posteriores así como los parches generados para las versiones anteriores, lo que sigue a continuación nos presenta una manera de cómo se nos presentan algunos factores de administración y técnicos para el mantenimiento de software, estos se agrupan según los tópicos siguientes:

- Factores técnicos.
- Factores administrativos.
- Estimación de costos.

CAPITULO 6. ADMINISTRACION DE LA CONFIGURACION DEL SOFTWARE

La administración de configuración es la disciplina encargada de identificar la configuración general de un sistema para así mantener su confiabilidad, adaptabilidad y configuración a los diferentes ciclos de vida. Está formalmente definida por la IEEE610.12-90 como “Disciplina aplicada de manera técnica y administrativa para la dirección y supervivencia para: Identificar y documentar las características físicas y funcionales de la configuración de los elementos, control en el cambio de sus características grabar y reportar cambios en el proceso de implementación, así como su estado y verificación del cumplimiento de sus requerimientos específicos”.

1. Administración de los procesos SCM

La SCM administra y controla la evolución e integridad del software, así como su verificación, control, reportes y configuración de la información. Una implementación exitosa del SCM requiere un cuidado especial y planeación y administración.

2. Identificación de la configuración del software

Identifica los elementos a ser controlados, establece e identifica esquemas y sus versiones, establece herramientas y técnicas utilizadas para administrar y controlar dichos elementos.

3. Control de la configuración del software

Le concierne la gestión de cambios durante el ciclo de vida, cubre los procesos que determinan los cambios a realizar, la autoridad para hacerlos y el soporte para la implementación de dichos cambios.

La información derivada de estas actividades es útil para medir el tráfico de cambios y ruptura de aspectos por rehacer.

4. Registro del estado de la configuración del Software

La contabilidad del estado de la configuración del software (SCSA) es la actividad de registrar y proporcionar la información necesaria para una gestión efectiva de la configuración del software.

5. Auditoría de la configuración de software

Es una actividad desarrollada independientemente para evaluar la conformidad de los productos de software, se encarga de aplicar regulaciones, estándares, planes de guía y procedimientos.

CAPITULO 7. GESTION DE LA INGENIERIA DEL SOFTWARE

Puede ser definida como las actividades de gestión de la aplicación, planeación, coordinación, medición, monitorización, control y reportes para asegurar el desarrollo y mantenimiento del software como sistemático, disciplinado y cuantificable.

Los aspectos de la gestión de la organización son importantes en términos del impacto en la ingeniería del software y en las políticas de gestión, esas políticas pueden ser influenciadas por los requerimientos de un software efectivo, mantenimiento y desarrollo.

1. Iniciación y Alcance

Se centra en la determinación eficaz de los requisitos del software por medio de varios métodos de inducción y la valoración de la viabilidad del proyecto desde distintos puntos de vista, una vez establecida la viabilidad, la tarea pendiente es la especificación de la validación de requisitos y del cambio de procedimientos.

2. Planificación de un Proyecto de Software

Está regulado por los alcances y los requisitos y por la viabilidad del proyecto, se evalúan los procesos de ciclo de vida y se selecciona el más apropiado.

3. Promulgación del proyecto de Software

Se ejecutan los planes y se divulgan los procesos incluidos en los planes, en este proceso hay total expectativa de la adhesión plena de los requisitos del contratista y el logro de los objetivos del proyecto, son actividades fundamentales para la promulgación la gestión, medición, supervisión, control e información del proyecto.

4. Revisión y Evaluación

Se evalúa el proceso global hacia el logro de los objetivos y satisfacción de los requisitos del contratista y se llevan a cabo valoraciones sobre la efectividad del proceso global hasta la fecha, del personal involucrado y de las herramientas y métodos utilizados.

5. Cierre

El proyecto llega a su fin cuando todos los planes y procesos implicados se han promulgado y completado, en esta fase se repasan ciertos criterios para el éxito del proyecto.

6. Medidas de la ingeniería del software

Aquí abordamos el tema de la medición en la ingeniería del software y su importancia para esto se sigue unas métricas y normas establecidas por entidades como la ISO y la IEEE.

Capítulo 8: proceso de ingeniería de software.

Los instrumentos de desarrollo de software son los instrumentos asistidos por ordenador que son requeridos para ayudar a los procesos de ciclo de vida de software. Los instrumentos permiten a acciones repetidas, bien definidas para ser automatizadas, reduciendo la carga cognoscitiva sobre el ingeniero de software que es entonces libre de concentrarse en los aspectos creativos del proceso.

Los instrumentos a menudo diseñados para apoyar el software particular métodos de la ingeniería, reduciendo cualquier carga administrativa asociada con la aplicación del método a mano. Como los métodos de la ingeniería de software, ellos son requeridos para hacer el software que trama mas sistemático, varían en el alcance de apoyar tareas individuales que abracan en el ciclo de vida completo.

Los métodos de la ingeniería de software imponen la estructura a la actividad de la ingeniería de software con el objetivo de hacer la actividad sistemática y en última instancia más probablemente de ser acertado. Los métodos por lo general proporcionan la notación y el vocabulario, procedimientos para realizar tareas identificables, y directrices para comprobar tanto el proceso como el producto. Ellos varían extensamente en el alcance, de una fase única del ciclo de vida al ciclo de vida completo. El énfasis en esta área de conocimiento esta sobre los métodos de la ingeniería de software que abarcan múltiples fases del ciclo de vida, ya que métodos específicos de fase son cubiertos por otras áreas de conocimiento.

Mientras hay manuales detallados sobre instrumentos específicos y numerosos papeles de investigación sobre instrumentos innovadores, escrituras genéricas técnicas sobre instrumentos de la ingeniería de software son relativamente escasas. Una dificultad es alta tarifa de cambio de instrumentos de software en general. Detalles específicos cambian con regularidad, haciendo difícil de proporcionar ejemplos concretos actualizarlos.

Capítulo 9: modelos y métodos de ingeniería de software.

El KA del proceso de ingeniería del software puede examinarse en dos niveles. El primer nivel engloba las actividades técnicas y de gestión dentro de los procesos del ciclo de vida del software realizadas durante la adquisición, desarrollo, mantenimiento y retirada del software.

El segundo es un meta-nivel, que se refiere a la definición, implementación, valoración, medición, gestión, cambios y mejoras de los procesos mismos del ciclo de vida del software. El primer nivel lo cubren las otras KAs en la Guía. Este KA se ocupa del segundo nivel.

El término “proceso de ingeniería del software” puede interpretarse de diversas maneras, y esto puede llevar a confusiones:

- Los estándares como IEEE hablan de procesos de ingeniería de software, lo que significa que hay muchos procesos involucrados, tales como procesos de desarrollo o proceso de configuración de gestión.
- Un segundo significado se refiere a una discusión general sobre procesos relacionados a la ingeniería del software. este es el significado que se pretende con el título de esta KA y el que se usa con mas frecuencia en la descripción de KA.
- Finalmente, un tercer significado podría referirse al conjunto actual de actividades realizadas dentro de una organización, que podría verse como un solo proceso, especialmente desde dentro de la organización, se utiliza este significado en el KA en muy pocos casos.

Los procesos de ingeniería de software tienen importancia no solo para las grandes organizaciones, más aun, las actividades relacionadas con los procesos pueden ser, y han sido, realizadas con éxito por pequeñas organizaciones, equipos e individuos.

Capítulo 10: Calidad de software.

La gestión de la ingeniería de software puede definirse como la aplicación para actividades de gestión – planificación, coordinación, mediciones, monitoreo, control e informes que asegure un desarrollo y mantenimiento del software sistemático, disciplinado y cuantificado.

El KA Gestión de Ingeniería de Software, por tanto, se encarga de la gestión y medición de la ingeniería del software. A pesar de que medir es un aspecto importante en todas las KA's no es hasta aquí que se presenta el tema de programas de medición.

Aunque por una parte sea verdad afirmar que, en cierto sentido, debiera ser posible gestionar la ingeniería del software de la misma manera que cualquier otro proceso existen aspectos específicos de los productos de software y de los procesos del ciclo de vida del software que complican una gestión efectiva- solo algunos cuales se apuntan a continuación:

- La percepción de los clientes es tal que con frecuencia existe una falta de aprecio de la complejidad inherente a la ingeniería de software, particularmente en relación al impacto que produce cambiar requisitos.
- Es casi inevitable que los propios procesos de ingeniería de software generen la necesidad de nuevos o modificados requisitos del cliente.
- La ingeniería de software incorpora necesariamente aspectos de creatividad y de disciplina – mantener un balance apropiado entre los dos es con frecuencia difícil.
- El grado de novedad y de complejidad del software son con frecuencia extremadamente altos.
- La tasa de cambio de la tecnología subyacente es muy rápida.

Con respecto a la ingeniería de software, las actividades de gestión tienen lugar en tres niveles: gestión organizacional y de infraestructura, gestión de proyectos, y programa de planificación y control de mediciones.

Capítulo 11: práctica profesional de ingeniería de software.

Para circunscribir la ingeniería de software, es necesario identificar las disciplinas con las que la ingeniería del software comparte un límite común. Este capítulo identificado, en orden alfabético, esas disciplinas relacionas. Por su puesto, las

disciplinas relacionadas también comparten varios limites en común entre ella mismos.

El informe borrador de volumen en la ingeniería de la computación de *Computing Curricula 2001 project(CC2001),* establece que *"*La ingeniería de la computación incorpora la ciencia y la tecnología del diseño, construcción, implementación y mantenimiento de los componentes software y hardware de los sistemas de cálculo moderno y del equipo controlado por ordenador.*"*

Este informe identifica las siguientes áreas de conocimiento (conocidas como áreas en el informe) para la ingeniería de la computación:

- Algoritmos y complejidad
- Arquitectura de ordenadores y organización
- Ingeniería de sistemas informáticos
- Circuitos y sistemas
- Lógica digital
- Estructuras directas
- Procesamiento de señales digitales
- Estructuras directas
- Fundamentos de programación
- Algoritmos y complejidad
- Arquitectura y organización
- Sistemas operativos
- Sistemas centralizados
- Lenguajes de programación
- Interacción Hombre-Maquina
- Gráficos y computación visual
- Sistemas inteligentes
- Gestión de información
- Problemas sociales y profesionales

- Ingeniería del software
- Ciencia computacional y calculo numérico

Capítulo 12: Economía de la ingeniería de software

La economía de la ingeniería de software se trata de tomar decisiones relacionadas con la ingeniería de software en un contexto empresarial. El éxito de un producto, servicio y solución de software depende de una buena gestión empresarial. Sin embargo, en muchas empresas y organizaciones, las relaciones comerciales de software con el desarrollo y la ingeniería de software siguen siendo vagas. Esta área de conocimiento (KA) proporciona una descripción general de la economía de la ingeniería de software. La economía es el estudio del valor, los costos, los recursos y su relación en un contexto o situación determinados. En la disciplina de la ingeniería de software, las actividades tienen costos, pero el software resultante en sí mismo tiene atributos económicos también. La economía de la ingeniería de software proporciona una forma de estudiar los atributos del software y los procesos de software de una manera sistemática que los relaciona con medidas económicas. Estas medidas económicas pueden sopesarse y analizarse al tomar decisiones que están dentro del alcance de una organización de software y aquellas dentro del alcance integrado de todo un negocio de producción o adquisición. La economía de la ingeniería de software se ocupa de alinear las decisiones técnicas de software con los objetivos comerciales de la organización. En todo tipo de organizaciones, ya sean "con fines de lucro", "sin fines de lucro" o gubernamentales, esto se traduce en una permanencia sostenible en el negocio. En las organizaciones "con fines de lucro", esto se relaciona además con el logro de un rendimiento tangible del capital invertido, tanto los activos como el capital empleado. Esta KA se ha formulado de manera que se dirija a todo tipo de organizaciones independientemente del enfoque, la cartera de productos y servicios, o la propiedad de capital y las restricciones impositivas. Decisiones como "¿Deberíamos utilizar un componente específico?" Puede parecer fácil desde una perspectiva técnica, pero puede tener serias implicaciones en la viabilidad comercial de un proyecto de software y el producto resultante. A menudo, los ingenieros se preguntan si tales preocupaciones se aplican en absoluto, ya que son "solo ingenieros". El análisis económico y la toma de decisiones son consideraciones de ingeniería importantes porque los ingenieros son capaces de evaluar decisiones tanto técnicamente como desde una perspectiva empresarial. Los contenidos de esta área de conocimiento son temas importantes que los ingenieros de software deben conocer, incluso si nunca están involucrados en decisiones comerciales concretas; Tendrán una visión completa de los problemas comerciales y el papel que juegan las consideraciones técnicas en la toma de decisiones comerciales. Muchas propuestas y decisiones de ingeniería, como hacer frente a comprar, tienen profundos impactos económicos intrínsecos que deben considerarse explícitamente. Este KA primero cubre los fundamentos, la terminología clave, los conceptos básicos y las prácticas comunes de la economía

de la ingeniería de software para indicar cómo la toma de decisiones en la ingeniería de software incluye, o debería incluir, una perspectiva empresarial. A continuación, proporciona una perspectiva del ciclo de vida, destaca la gestión de riesgos e incertidumbres y muestra cómo se utilizan los métodos de análisis económico. Algunas consideraciones prácticas finalizan el área de conocimiento. destaca la gestión de riesgos e incertidumbres, y muestra cómo se utilizan los métodos de análisis económico. Algunas consideraciones prácticas finalizan el área de conocimiento. destaca la gestión de riesgos e incertidumbres, y muestra cómo se utilizan los métodos de análisis económico. Algunas consideraciones prácticas finalizan el área de conocimiento.

Capítulo 13: Fundamentos de la computación

El alcance del área de conocimiento de Fundamentos de la Computación (KA) abarca el entorno operativo y de desarrollo en el que el software evoluciona y se ejecuta. Debido a que ningún software puede existir en el vacío o ejecutarse sin una computadora, el núcleo de dicho entorno es la computadora y sus diversos componentes. El conocimiento sobre la computadora y sus principios subyacentes de hardware y software sirve como marco sobre el que se ancla la ingeniería de software. Por lo tanto, todos los ingenieros de software deben tener un buen conocimiento de Computing Foundations KA.

En general, se acepta que la ingeniería de software se basa en la informática. Por ejemplo, “Ingeniería de software 2004: Pautas curriculares para programas de licenciatura en ingeniería de software” establece claramente: “Un aspecto particularmente importante es que la ingeniería de software se basa en la informática y las matemáticas” (cursiva agregada).

Steve Tockey escribió en su libro Return on Software : Tanto la informática como la ingeniería de software se ocupan de las computadoras, la informática y el software. La ciencia de la computación, como cuerpo de conocimiento, es el núcleo de ambos. La ingeniería de software se ocupa de la aplicación de computadoras, computación y software para propósitos prácticos, específicamente el diseño, construcción y operación de sistemas de software eficientes y económicos. Por lo tanto, en el núcleo de la ingeniería de software se encuentra la comprensión de la informática.

Si bien pocas personas negarán el papel que desempeña la informática en el desarrollo de la ingeniería de software como disciplina y como cuerpo de conocimiento, no se puede exagerar la importancia de la informática para la ingeniería de software; por lo tanto, este KA de Fundamentos de Computación se está escribiendo.

La mayoría de los temas discutidos en Computing Foundations KA también son temas de discusión en cursos básicos impartidos en programas de pregrado y posgrado en ciencias de la computación. Dichos cursos incluyen programación, estructura de datos, algoritmos, organización de computadoras, sistemas operativos, compiladores, bases de datos, redes, sistemas distribuidos, etc. Por lo tanto, al desglosar temas, puede ser tentador descomponer el KA de Fundamentos de Computación de acuerdo con estas divisiones que se encuentran a menudo en los cursos relevantes.

Sin embargo, una división de temas puramente basada en cursos adolece de serios inconvenientes. Por un lado, no todos los cursos de informática están relacionados o son igualmente importantes con la ingeniería de software. Por lo tanto, algunos temas que de otro modo se cubrirían en un curso de informática no se tratan en esta KA. Por ejemplo, gráficos por computadora, aunque es un curso importante en un programa de grado en ciencias de la computación, no se incluye en esta KA.

En segundo lugar, algunos temas discutidos en esta guía no existen como cursos independientes en programas de ciencias de la computación de pregrado o posgrado. En consecuencia, es posible que dichos temas no se cubran adecuadamente en un desglose puramente basado en cursos. Por ejemplo, la abstracción es un tema incorporado en varios cursos de informática diferentes; No está claro a qué curso debe pertenecer la abstracción en un desglose de temas basado en cursos.

The Computing Foundations KA se divide en diecisiete temas diferentes. La utilidad directa de un tema para los ingenieros de software es el criterio utilizado para seleccionar los temas que se incluirán en esta KA. La ventaja de este desglose basado en temas es que se basa en la creencia de que Computing Foundations, si se quiere comprender con firmeza, debe considerarse como una colección de temas conectados lógicamente que sustentan la ingeniería de software en general y la construcción de software en particular.

El KA de Fundamentos de Computación está estrechamente relacionado con los KA de Diseño de Software, Construcción de Software, Pruebas de Software, Mantenimiento de Software, Calidad de Software y Fundamentos Matemáticos.

Capítulo 14: Fundamentos matemáticos

Los profesionales del software conviven con los programas. En un lenguaje muy simple, uno puede programar solo para algo que sigue una lógica bien entendida y no ambigua. El área de conocimiento de Fundamentos matemáticos (KA) ayuda a los ingenieros de software a comprender esta lógica, que a su vez se traduce al código del lenguaje de programación. Las matemáticas que son el enfoque principal

en este KA son bastante diferentes de la aritmética típica, donde los números se tratan y se discuten. La lógica y el razonamiento son la esencia de las matemáticas que un ingeniero de software debe abordar.

Las matemáticas, en cierto sentido, son el estudio de sistemas formales. La palabra “formal” está asociada a la precisión, por lo que no puede haber una interpretación ambigua o errónea del hecho. Por lo tanto, las matemáticas son el estudio de todas y cada una de las verdades ciertas sobre cualquier concepto. Este concepto puede ser tanto de números como de símbolos, imágenes, sonidos, videos, casi cualquier cosa. En resumen, no solo los números y las ecuaciones numéricas están sujetos a precisión. Por el contrario, un ingeniero de software necesita tener una abstracción precisa en un dominio de aplicación diverso.

La guía SWEBOK Fundamentos matemáticos KA cubre técnicas básicas para identificar un conjunto de reglas para el razonamiento en el contexto del sistema en estudio. Todo lo que se pueda deducir siguiendo estas reglas es una certeza absoluta dentro del contexto de ese sistema. En este KA, se definen y discuten técnicas que pueden representar y hacer avanzar el razonamiento y juicio de un ingeniero de software de una manera precisa (y por lo tanto matemática). El lenguaje y los métodos de lógica que se discuten aquí nos permiten describir pruebas matemáticas para inferir de manera concluyente la verdad absoluta de ciertos conceptos más allá de los números. En resumen, puede escribir un programa para un problema solo si sigue alguna lógica. El objetivo de este KA es ayudarlo a desarrollar la habilidad para identificar y describir dicha lógica.

Capítulo 15: Fundamentos de ingeniería

Un método de ingeniería para la resolución de problemas implica proponer soluciones o modelos de soluciones y luego realizar experimentos o pruebas para estudiar las soluciones o modelos propuestos. Por lo tanto, los ingenieros deben comprender cómo crear un experimento y luego analizar los resultados del experimento para evaluar la solución propuesta. Los métodos empíricos y las técnicas experimentales ayudan al ingeniero a describir y comprender la variabilidad en sus observaciones, identificar las fuentes de variabilidad y tomar decisiones.

Para cumplir con sus responsabilidades, los ingenieros deben comprender cómo varían las diferentes características de los productos y procesos. Los ingenieros a menudo se encuentran con situaciones en las que es necesario estudiar la relación entre diferentes variables. Un punto importante a tener en cuenta es que la mayoría de los estudios se llevan a cabo sobre la base de muestras, por lo que los resultados observados deben comprenderse con respecto a la población completa.

Por lo tanto, los ingenieros deben desarrollar una comprensión adecuada de las técnicas estadísticas para recopilar datos confiables en términos de muestreo y análisis para llegar a resultados que puedan generalizarse. Estas técnicas se analizan a continuación.

Saber qué medir y qué método de medición utilizar es fundamental en los esfuerzos de ingeniería. Es importante que todos los involucrados en un proyecto de ingeniería comprendan los métodos de medición y los resultados de medición que se utilizarán.

Las mediciones pueden ser físicas, ambientales, económicas, operativas o algún otro tipo de medición que sea significativa para el proyecto en particular. Esta sección explora la teoría de la medición y cómo es fundamental para la ingeniería.

IEEE define la ingeniería como “la aplicación de un enfoque sistemático, disciplinado y cuantificable a estructuras, máquinas, productos, sistemas o procesos”. Este capítulo describe algunas de las habilidades y técnicas fundamentales de ingeniería que son útiles para un ingeniero de software. La atención se centra en temas que apoyan a otros KA mientras se minimiza la duplicación de temas cubiertos en otras partes de este documento.

A medida que madura la teoría y la práctica de la ingeniería de software, es cada vez más evidente que la ingeniería de software es una disciplina de ingeniería que se basa en conocimientos y habilidades comunes a todas las disciplinas de ingeniería. Esta área de conocimiento de Fundamentos de Ingeniería (KA) se ocupa de los fundamentos de ingeniería que se aplican a la ingeniería de software y otras disciplinas de ingeniería. Los temas de esta KA incluyen métodos empíricos y técnicas experimentales; análisis estadístico; medición; diseño de ingeniería; modelado, creación de prototipos y simulación; estándares; y análisis de la causa raíz. La aplicación de este conocimiento, según corresponda, permitirá a los ingenieros de software desarrollar y mantener el software de manera más eficiente y eficaz. Completar su trabajo de ingeniería de manera eficiente y efectiva es un objetivo de todos los ingenieros en todas las disciplinas de la ingeniería.